User Guide for the SVM Toolbox da Rouen*

Machine Learning Team da Rouen, LITIS - INSA Rouen†

May 25, 2009

Contents

*This work has been partially funded by the ANR project KernSig
†http://sites.google.com/site/kernsig/sofware-de-kernsig

1 Introduction

Kernel methods are hot topics in machine learning communicity. Far from being panaceas, they yet represent powerful techniques for general (nonlinear) classification and regression. The SVM-KM toolbox is a library of MATLAB routies for support vector machine analysis. It is fully written in Matlab (even the QP solver). It includes:

- SVM Classification (standard, nearest point algorithm)
- Multiclass SVM (one against all, one against one and M-SVM)
- Large scale SVM classification
- SVM epsilon and nu regression
- One-class SVM
- Regularization networks
- SVM bounds (Span estimate, radius/margin)
- Wavelet Kernel
- SVM Based Feature Selection
- Kernel PCA
- Kernel Discriminant Analysis
- SVM AUC Optimization and RankBoost
- Kernel Basis Pursuit and Least Angle Regression Algorithm
- Wavelet Kernel Regression with backfitting
- Interface with a version of libsvm

2 Installation procedure

Preparation

1. download the packages from:
   $http://asi.insa-rouen.fr/enseignants/\ arakotom/toolbox/index.html$
2. to make all the functions available, download another toolbox - WAVELAB from:
   $http://www-stat.stanford.edu/\ wavelab/Wavelab_850/download.html$

Installation steps

1. Extract the .zip file into an appropriate directory ***.
2. Set path for the SVM-KM toolbox, you can finish it by 'File/Set Path/Add Folder/***', then click Save.

If you have a successful installation, when you type 'TestSaintyCheck' in Command Window, all the functions in the SVM-KM toolbox will be checked, and you will see something like this at the end:
'Positive definite Sanity Check : OK'.

3 Support Vector Classification

3.1 Principle

Training set is denoted as $(x_i, y_i)_{i=1,\cdots,n}$, x_i is a input vector, y_i is the label of it, and n is the number of training data, then a classification model of SVM in a high-dimension feature space can be represented as follows:

$$f(x) = \text{sgn}\left(\langle w, \phi(x)\rangle + b\right) \tag{1}$$

where, w is a weight vector, b is a bias, $\phi(x)$ is a nonlinear mapping from the input variables into a high dimension feature space $<,>$ denotes the dot product.

The optimal classification hyperplane can be obtained by the following primal formulation :

$$\begin{array}{lll} \min_{w,b,\xi} & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i & \\ \text{s.t.} & y_i\left(\langle w, \phi(x_i)\rangle + b\right) \geq 1 - \xi_i & i = 1, \cdots, n \\ & \xi_i \geq 0 & i = 1, \cdots, n \end{array} \tag{2}$$

where $\frac{1}{2}\|w\|^2$ controls the complexity of the model, ξ_i is a slack variable measuring the error on x_i, C is a regularization parameter, which determines the trade off between the empirical error and the complexity of the model. A Lagrangian function corresponding to (2) can be described as follows by introducing Lagrange multipliers $\alpha_i \geq 0$ and $\gamma_i \geq 0$.

$$\mathcal{L} = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left[y_i(\langle w, \phi(x_i)\rangle + b) - 1 + \xi_i\right] - \sum_{i=1}^{n}\gamma_i\xi_i \tag{3}$$

Based on Karush-Kuhn-Tucker (KKT) optimality conditions:

$$\begin{array}{l} \nabla_w L = 0 \\ \nabla_b L = 0 \\ \nabla_{\xi_i} L = 0 \end{array}$$

we can get the following relations:

$$\begin{aligned} w &= \sum_{i=1}^{n}\alpha_i y_i \phi(x_i) \\ \sum_{i=1}^{n}\alpha_i y_i &= 0 \end{aligned}$$

and the box constraints

$$0 \leq \alpha_i \leq C, \qquad \forall i = 1, \cdots, n.$$

Plugging the later equations in the Lagrangian, we derive the formulation of the dual problem. Hence the SVM classification becomes a Quadratic Programming (QP) problem:

$$\min_{\alpha_i, i=1,\cdots,n} \quad \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j k(x_i, x_j) - \sum_{i=1}^{n}\alpha_i \tag{4}$$

$$\text{s.t.} \quad \sum_{i=1}^{n}\alpha_i y_i = 0$$

$$0 \leq \alpha_i \leq C, \qquad \forall i = 1, \cdots, n. \tag{5}$$

From the solution of the dual, only the data that corresponding to non-zero values of α_i work, and they were called support vectors (SV). So (1) can be represented as:

$$f(x) = \text{sgn}\left(\sum_{i \in SV} \alpha_i y_i \kappa(x_i, x) + b\right) \tag{6}$$

where $\kappa(x_i, x) =< \phi(x_i), \phi(x) >$ is the kernel function.

3.2 How to solve the classification task using the toolbox?

Mentioned above is a binary classification task, it can be solved by seeking solutions of a quadratic programming (QP) problem. In the SVM-KM toolbox, `monqp` and `monqpCinfty` are QP solvers. They are internal functions which can be called by external functions. The only thing you need to do is to provide conditioning parameters for them.

`svmclass` is the corresponding function to solve the binary classification task. For simplicity, you can launch the script m-file *exclass* to obtain a more detailed information.

The «checkers» data used for this example are generated by the lines below. The main tool is the function `dataset` which compiles a certain number of toy problems (see xxrefxxtoxxfctxxdataset for more details).

```
nbapp=200;
nbtest=0;
sigma=2;
[xapp,yapp]=dataset('Checkers',nbapp,nbtest,sigma);
```

The decision function is learned via this segment of code lines:

```
c = inf;
epsilon = .000001;
kerneloption= 1;
kernel='gaussian';
verbose = 1;
tic
[xsup,w,b,pos]=svmclass(xapp,yapp,c,epsilon,kernel,
   kerneloption,verbose);
```

The regularisation parameter C is set to 1000. A gaussian kernel with bandwidth $\sigma = 0.3$ is used. By setting the parameter `verbose`, the successive iterations of the dual resolution are monitored. To ensure the well conditioning of the linear system (with bound constraints on α_i) induced by the QP problem, a ridge regularization with parameter λ is used. Typical value of λ is 10^{-7}. The dual resolution is based on the active constraints procedure ref xxx.

As a result we obtain the parameters vector w with $w_i = \alpha_i y_i$, the bias b of the decision function $f(x)$. The function yields also the index `pos` of the points which are support vectors among the training data.

The evaluation of the decision function on testing data generated as a mesh grid of the training data supsbapce is illustrated by the lines below.

```
%--------------Testing Generalization performance
   ---------------
[xtesta1,xtesta2]=meshgrid([-4:0.1:4],[-4:0.1:4]);
[na,nb]=size(xtesta1);
```

Figure 1: Illustration of the nonlinear Gaussian SVM using different kernel parameters: *kerneloption* = 0.3 (on the left) and *kerneloption* = 0.8 (on the right) on 'checkers' data set. (other parameters: n = 500; sigma=1.4; lambda = 1e-7; C = 1000; kernel='gaussian')

```
xtest1=reshape(xtesta1,1,na*nb);
xtest2=reshape(xtesta2,1,na*nb);
xtest=[xtest1;xtest2]';
```

Figure 1 shows results of nonlinear Gaussian SVM using different kernel parameters.

From the results, you can see: when kernel parameter is larger, the decision function is more flater. Figure 2 shows results of a nonlinear SVM using different kernel type to solve the classification task for a classical dataset 'Gaussian' (with two gaussianly distributed classes with similar variances but different means).

3.3 Extensions of binary support vector machine

3.3.1 ν-SVM

The ν-SVM has been derived so that the soft margin has to lie in the range of zero and one.The parameter ν is not controlling the trade off between the training error and the generalization error; instead it now has two roles:
1. an upper bound on the fraction of margin errors.
2. the lower bound on the fraction of support vectors.
The ν-SVM formulation is as follows:

$$\begin{array}{ll} \min \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{n}\sum_{i=1}^{n}\xi_i & \\ \text{s.t.} \quad y_i\left(\langle w, \phi(x_i)\rangle + b\right) \geq \rho - \xi_i & i = 1, \cdots, n \\ \xi_i \geq 0 & i = 1, \cdots, n \\ \rho \geq 0 & \end{array} \tag{7}$$

ν is a user chosen parameter between 0 and 1.
We consider the Lagrangian:

$$\mathcal{L} = \frac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{n}\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i\left[y_i(\langle w, \phi(x_i)\rangle + b) - \rho + \xi_i\right] - \sum_{i=1}^{n}\beta_i\xi_i - \delta\rho$$
$$\text{with} \qquad \delta \geq 0, \quad \alpha_i \geq 0, \quad \beta_i \geq 0, \qquad i = 1, \cdots, n$$

Figure 2: Comparing Gaussian and polynomial Kernels: on the left is final decision function using polynomial kernel(lambda = 1e-10; C = 1; kernel='poly'; kerneloption=1;): on the right is final decision function using Gaussian kernel(kernel='gaussian'; kerneloption=0.25;)

Figure 3: Comparing SVM learning with initialization of Lagrangian Multipliers (on the right) and without initialization (on the left)

Figure 3 shows that with initialization of Lagrangian Multipliers you can obtain more flatter decision function.

Based on KKT conditions, we obtained the following equations:

$$\begin{aligned} w &= \sum_{i=1}^{n} \alpha_i y_i \phi(x_i) \\ \sum_{i=1}^{n} \alpha_i y_i &= 0 \end{aligned}$$

and the following inequality constraints

$$\begin{aligned} \sum_{i=1}^{n} \alpha_i &\geq \nu \\ 0 \leq \alpha_i &\leq \frac{1}{n}, \quad i = 1, \cdots, n \\ 0 \leq \beta_i &\leq \frac{1}{n}, \quad i = 1, \cdots, n \end{aligned}$$

The dual problem which is solved is derived as:

$$\begin{array}{rc} \min\limits_{\alpha_i, i=1,\cdots,n} & \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j k(x_i, x_j) \\ \text{s.t.} & \sum_{i=1}^{n} \alpha_i y_i = 0 \\ & \sum_{i=1}^{n} \alpha_i = \nu \\ & 0 \leq \alpha_i \leq \frac{1}{n}, \qquad \forall i = 1, \cdots, n. \end{array} \tag{8}$$

The data used for this example are generated by the lines below which use the function function `dataset`.

```
n = 100;

sigma=0.4;
[Xapp,yapp,xtest,ytest]=dataset('gaussian',n,0,sigma);
[Xapp]=normalizemeanstd(Xapp);
```

The decision function is learned +via this segment of code lines:

```
 kernel='gaussian';
 kerneloption=4;
 nu=0.1;
lambda = 1e-12;


[xsup,w,w0,rho,pos,tps,alpha] = svmnuclass(Xapp,yapp,nu,
   lambda,kernel,kerneloption,1);
```

Here the ν parameter was set to 0.1. The function returns the paameters $w_i = \alpha_i y_i$ in the vector w, the bias b, the margin parameter ρ as well as the support vectors `xsup` and their corresponding indexes `pos` in the training data.

The evaluation of the solution on testing data is achieved when launching these lines

```
ypredapp = svmval(Xapp,xsup,w,w0,kernel,kerneloption,1);

%------- Building a 2D Grid for function evaluation
   -------------------------
```

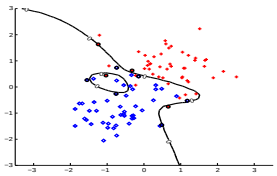

Figure 4: Illustration of ν-SVM

```
[xtest1 xtest2]  = meshgrid([-1:.05:1]*3.5,[-1:0.05:1]*3)
    ;
nn = length(xtest1);
Xtest = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn
    ,1)];
```

Figure 4 shows results of ν-SVM to classify the 'Gaussian' dataset.

3.3.2 Linear Programming SVM

Compared with standard SVM, the decision function in the linear programming SVM is as follows:

$$f(x) = \sum_{i=1}^{n} \alpha_i \kappa(x, x_i) + b \tag{9}$$

where α_i take on real values. The kernel κ need not be positive semi-definite.
We consider minimizing:

$$\min L(\alpha, \xi) = \sum_{i=1}^{n} (|\alpha_i| + C\xi_i) \tag{10}$$

Subject to:

$$\sum_{i=1}^{n} \alpha_i \kappa(x_j, x_i) + b \geq (1 - \xi_i), for j = 1, \cdots, n$$

where ξ_i are positive slack variables and C is a margin parameter.
The data for the example was generated by the codes below:

```
nbapp=100;
nn=20;
nbtest=nn*nn;
sigma=0.5;
[xapp,yapp,xtest,ytest,xtest1,xtest2]=dataset('gaussian',
    nbapp,nbtest,sigma);
```

Learning parameters are obtained via the follows lines:

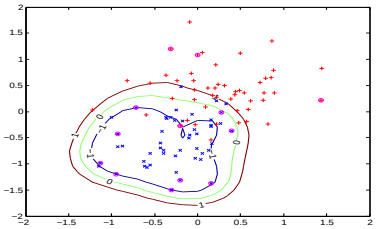

Figure 5: Illustration of Classification task using LPSVM

```
C = 20;  lambda = 0.000001;
kerneloption = 0.5;
kernel='gaussian';
verbose=1;
[xsup,w,b,pos]=LPsvmclass(xapp,yapp,C,lambda,kernel,
    kerneloption);
```

Testing data are generated by:

```
[xtest1 xtest2]  = meshgrid([-1:.05:1]*3.5,[-1:0.05:1]*3)
    ;
nn = length(xtest1);
xtest = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn
    ,1)];
```

Predicted value by this LPSVM is obtained:

```
ypred = reshape(ypred,nn,nn);
```

Figure 5 shows result of the example.

3.3.3 Density level estimation using oneclass SVM

One class SVM seeks a hypersphere that contains target class examples as many as possible while keeps its radius small. Through minimizing this hypersphere's volume we hope to minimize chance of accepting outliers.

$$\begin{array}{lll} \min \frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho & & \\ \text{s.t.} \quad \langle w, \phi(x_i)\rangle \geq \rho - \xi_i & i = 1, \cdots, n & \\ \xi_i \geq 0 & i = 1, \cdots, n & \end{array} \tag{11}$$

where $0 \leq \nu \leq 1$ is a user chosen parameter.
Consider the Lagrangian:

$$\mathcal{L} = \frac{1}{2}\|w\|^2 + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i - \rho - \sum_{i=1}^{n}\alpha_i\left[\langle w, \phi(x_i)\rangle - \rho - \xi_i\right] - \sum_{i=1}^{n}\gamma_i\xi_i \qquad (12)$$

Based on KKT conditions, we can obtain:

$$\begin{aligned} w &= \sum_{i=1}^{n}\alpha_i\phi(x_i) \\ \sum_{i=1}^{n}\alpha_i &= 1 \end{aligned}$$

together with the box constraints

$$0 \leq \alpha_i \leq \frac{1}{\nu n}, \quad \forall\ i = 1, \cdots, n$$

Then, final decision function is:

$$f(x) = \sum_{i=1}^{n}\alpha_i\left\langle\phi(x_i), \phi(x)\right\rangle - \rho = \sum_{i=1}^{n}\alpha_i k(x_i, x) - \rho \qquad (13)$$

The parameters α_i are derived by solving the dual problem

$$\begin{aligned} \min_{\alpha_1,\cdots,\alpha_n} \quad & \tfrac{1}{2}\textstyle\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j k(x_i, x_j) \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{n}\alpha_i = 1 \\ & 0 \leq \alpha_i \leq \tfrac{1}{\nu n}, \quad \forall\ i = 1, \cdots, n \end{aligned}$$

Once this problem is solved for some ν yielding the α_i, the offset parameter ρ is computed from KKT conditions.

The data used for this example are generated by the lines below which use the function function `dataset`.

```
n=100;
sigma=0.2;
[xapp,yapp]=dataset('gaussian',n,0,sigma);
```

The decision function is learned +via this segment of code lines:

```
nu=0.1;
kernel='gaussian';
kerneloption=0.33;
verbose=1;
[xsup,alpha,rho,pos]=svmoneclass(xapp,kernel,kerneloption
    ,nu,verbose);
```

Here the ν parameter was set to 0.1. The function returns the parameters $w_i = \alpha_i$ in the vector w, the bias b, the margin parameter ρ as well as the support vectors `xsup` and their corresponding indexes `pos` in the training data.

The evaluation of the solution on testing data is achieved when launching these lines

```
[xtest,xtest1,xtest2,nn]=DataGrid2D
    ([-3:0.2:3],[-3:0.2:3]);
ypred=svmoneclassval(xtest,xsup,alpha,rho,kernel,
    kerneloption);
ypred=reshape(ypred,nn,nn);
```

3.3.4 svmclassnpa

Main ROUTINE For Nearest Point Algorithm.
The basic problem treated by *svmclassnpa* is one that does not allow classification violations. The problem is converted to a problem of computing the nearest point between two convex polytopes. For problems which require classification violations to be allowed, the violations are quadratically penalized and they are converted into problems in which there are no classification violations.
This function solve the SVM Classifier problem by transforming the soft-margin problem into a hard margin problem. The problem is equivalent to look for the nearest points of two convex hulls. This is solved by the point of minimum norm is the Minkowski set $(U - V)$ where U is the class 1 set and V is the class 2 set. For more detailed information, please find:
S. S. Keerthi, S. K. Shevade, C. Bhattacharyya, and K. R. K. Murthy. A Fast Iterative Nearest Point Algorithm for Support Vector Machine Classifier Design, *IEEE TRANSACTIONS ON NEURAL NETWORKS*, VOL. 11, NO. 1, JANUARY 2000

The data used for this example are generated by the lines below which use the function function `dataset`.

```
N=50;
sigma=0.3;
[xapp,yapp,xtest,ytest]=dataset('gaussian',N,N,sigma);
```

The decision function is learned +via this segment of code lines:

```
c =1e6;
%c1=10000;
c2=100;
c1=10000;
epsilon = .000001;
gamma =[0.2 0.2 100];
gamma1=0.4;
kernel='gaussian';
verbose = 1;
[xsup1,w1,b1,pos1]=svmclassnpa(xapp,yapp,c1,kernel,gamma1
   ,verbose);
```

Here the parameter $c1$ was set to 10000, it caused a slightly modification of kernel using the following equation:
$\kappa^*(x_k, x_l) = \kappa(x_k, x_l) + \frac{1}{c1}\delta_{kl}$
where δ_{kl} is one if $k = l$ and zero otherwise.

Here the ν parameter was set to 0.1. The function returns the parameters $w_i = \alpha_i$ in the vector w, the bias b, the margin parameter ρ as well as the support vectors `xsup` and their corresponding indexes `pos` in the training data.

The evaluation of the solution on testing data is achieved when launching these lines

```
ypred1=svmval(xtest,xsup1,w1,b1,kernel,gamma1);

%
   ------------------------------------------------------------

na=sqrt(length(xtest));
```

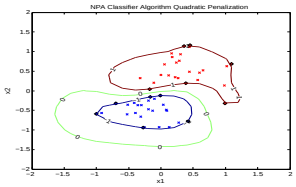

Figure 6: Illustration of SVM for Nearest Point Algorithm with Quadratic Penalization

```
ypredmat1=reshape(ypred1,na,na);
xtesta1=reshape(xtest(:,1),na,na);
```

Figure 6 shows results of SVM using Nearest Point Algorithm to classify the 'Gaussian' dataset. The green line denotes the decision function.

3.3.5 Semiparametric SVM

A common semipqrametric method is to fit the data with the parametric model and train the nonparametric add-on on the errors of the parametric part, i.e. fit the nonparametric part to the errors. One can show that this is useful only in a very restricted situation. The decision function is as follows:

$$f(x) = \langle w, \phi(x) \rangle + \sum_{i=1}^{m} \beta_i \phi_i(x) \tag{14}$$

Note that this is not so much different from the original setting if we set $n = 1$ and $\phi_1(.) = 1$. In the toolbox, if the span matrix for semiparametric learning is given, the function *svmclass* can solve semiparametric SVM problem.

The data used for example are generated by the lines below which use the function `dataset`.

```
nbapp=500;
nn=20;
nbtest=nn*nn;
sigma=0.1;
[xapp,yapp,xtest,ytest,xtest1, xtest2]=dataset('CosExp',
   nbapp,nbtest,sigma);
```

Learning parameters are set by the following code lines:

```
lambda = 1e-7;
C = 100;
kernel='gaussian';
kerneloption=1;
```

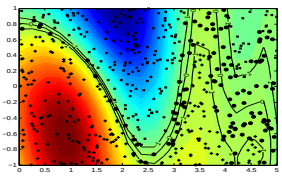

Figure 7: Illustration of Semiparametric SVM

Span matrix is set when launching these lines:

```
phi = phispan(xapp,'sin2D');
phitest = phispan(xtest,'sin2D');
```

Here, we set the type of parametric functions as 'sin2D'.

Finally, the decision function and the evaluation of the method are achieved:

```
[xsup,w,w0,tps,alpha] = svmclass(xapp,yapp,C,lambda,
   kernel,kerneloption,1,phi);
[ypred,y1,y2] = svmval(xtest,xsup,w,w0,kernel,
   kerneloption,phitest,1);
```

Figure 7 shows results of semiparatric SVM to classify the data set 'CosExp' (estabished by the function $sign(0.95 * cos(0.5 * (exp(x) - 1)))$) which is not a easy task.

3.3.6 Wavelet SVM

The goal of the Wavelet SVM is to find the optimal approximtion or classification in the space spanned by multidimensional wavelets or wavelet kernels. The idea behind the wavelet analysis is to express or approximate a signal or function by a family of functions generated by $h(x)$ called the mother wavelet:

$$h_{a,c}(x) = |a|^{-1/2} h\left(\frac{x-c}{a}\right)$$

where $x, a, c \in R$. a is a dilation factor, and c is a translation factor. Therefore the wavelet transform of a function $f(x)$ can be written as:

$$W_{a,c}(f) = \langle f(x), h_{a,c}(x) \rangle$$

For a mother kernel, it is necessary to satisfy mentioned above conditions.
For a common multidimensional wavelet function, we can write it as the product of one-dimensional (1-D) wavelet functions:

$$h(x) = \prod_{i=1}^{N} h(x_i)$$

where N is the dimension number.
Let $h(x)$ be a mother kernel, then dot-product wavelet kernels are:

$$\kappa(x, x^{'}) = \prod_{i=1}^{N} h\left(\frac{x_i - c_i}{a}\right) h\left(\frac{x_i^{'} - c_i^{'}}{a}\right)$$

Then the decision function for classification is:

$$f(x) = \text{sgn}\left(\sum_{i=1}^{n} \alpha_i y_i \prod_{j=1}^{N} h\left(\frac{x^j - x_i^j}{a_i}\right) + b\right) \tag{15}$$

The data used for this example are generate by the following lines:

```
n = 160;
ntest=400;
sigma=1.2;
[Xapp,yapp,xtest,ytest, xtest1, xtest2]=dataset('Checkers
    ',n,ntest,sigma);
```

Kernel options are set when lanching the following lines:

```
kernel='tensorwavkernel';


kerneloption.wname='Haar';    % Type of mother wavelet
kerneloption.pow=8;           % number of dyadic
    decomposition
kerneloption.par=4;           % number of vanishing
    moments
kerneloption.jmax=2;
kerneloption.jmin=-2;
kerneloption.father='on';    % use father wavelet in
    kernel
kerneloption.coeffj=1/sqrt(2); % coefficients of
    ponderation of a given j scale
```

Here q 2D Tensor-Product Kernel are used. To evaluate the Wavelet SVM, a Gaussian SVM is introduced:

```
kernel2='gaussian';
kernel2option=0.5;
```

The SVM calculation process is achieved by the following code:

```
[xsup,w,w0,tps,alpha] = svmclass(Xapp,yapp,C,lambda,
    kernel,kerneloption,1);
tpstensor=toc;
ypred = svmval(Xtest,xsup,w,w0,kernel,kerneloption,[ones(
    length(Xtest),1)]);
ypred = reshape(ypred,nn,nn);
```

Figure 8 shows that: both the Wavelet kernel and Gaussian kernel can achieve good performance, figure 9 shows that the Wavelet SVM achieves better performance in some region.

Figure 8: Illustration of Wavelet SVM and Gaussian SVM

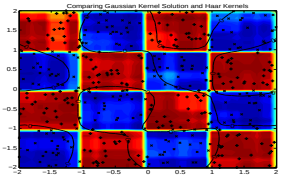

Figure 9: Comparing Gaussian kernel solutions and Wavelet kernels

3.3.7 Large Scale SVM

SVM algorithm for large scale using decomposition algorithm. The whole training set is divided into many subset which is much smaller. When training, remove the non support vectors from the working set and fill the chunk up with the data the current estimator would make errors on to form new working set. Then retrain the system and keep on iterating until after training the KKT-conditions are satisfied for all samples.

The data used for this large scale classification task are generated by the lines below which use the function function `dataset`.

```
n = 7000;
nbtest=1600;
sigma=0.3;  % This problem needs around a thousand
    iteration
            % you can speed up the example by setting
                sigma=0.3 for instance
[Xapp,yapp]=dataset('gaussian',n,0,sigma);
```

Here the size of training set is 7000. The decision function is learned +via this segment of code lines:

```
lambda = 1e-10;
C = 10;
kernel='gaussian';
kerneloption=1;
qpsize=300;
chunksize=300;
verbose=1;
span=[];
[xsup,w,w0,pos,tps,alpha] = svmclassLS(Xapp,yapp,C,lambda
    ,kernel,kerneloption,verbose,span,qpsize,chunksize);
```

The *chunksize* is set to 300, which is much smaller than that of working set. To evaluate the solution on testing data is achieved when lanching the following lines.

```
[xtest1 xtest2]  = meshgrid(linspace(-4,4,sqrt(nbtest)));
nn = length(xtest1);
Xtest = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn
    ,1)];



%-------------- Evaluating the decision function
ypred = svmval(Xtest,xsup,w,w0,kernel,kerneloption,[ones(
    length(Xtest),1)]);
ypred = reshape(ypred,nn,nn);
```

Figure 10 shows results of SVM algorithm for large scale using decomposition method to classify the 'Gaussian' dataset.

3.3.8 svmclassL2LS

Large scale SVM algorithm with quadratic penalty. The data used for this example are generated by the lines below which use the function function `dataset`.

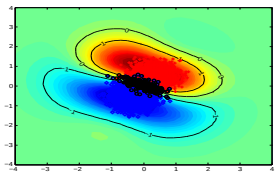

Figure 10: Illustration of SVM algorithm for large scale using decomposition method

```
n = 500;

sigma=0.3;
[Xapp,yapp,xtest,ytest]=dataset('gaussian',n,0,sigma);
```

The decision function is learned +via this segment of code lines:

```
lambda = 1e-12;
C = 1;
 kernel='poly';
 kerneloption=2;
 qpsize=10;
 verbose=1;
 span=1;
[xsup,w,w0,pos,tps,alpha] = svmclassL2LS(Xapp,yapp,C,
   lambda,kernel,kerneloption,verbose,span,qpsize);
[xsup1,w1,w01,pos1,tps,alpha1] = svmclassL2(Xapp,yapp,C,
   lambda,kernel,kerneloption,verbose,span);
```

The following lines are introduced to evaluate the decision function:

```
ypred = svmval(Xtest,xsup,w,w0,kernel,kerneloption,[ones(
   length(Xtest),1)]);
ypred = reshape(ypred,nn,nn);
 ypred1 = svmval(Xtest,xsup1,w1,w01,kernel,kerneloption,[
    ones(length(Xtest),1)]);
ypred1 = reshape(ypred1,nn,nn);
```

Figure fig:exsvmlsl2 shows results of SVM algorithm with quadratic penalization.

3.3.9 Multi-class SVM

One against All (OAA) algorithm and One Against One (OAO) are popular algorithms to solve multi-classification task. OAA SVM required unanimity among all SVM: a data point would be classified under a certain class if and only if that class's SVM accepted it and all other classes' SVMs rejected it, and OAO SVM approach

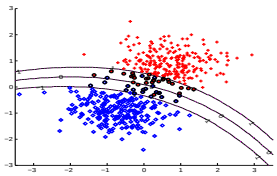

Figure 11: Illustration of SVM algorithm using decomposition method with quadratic penalization

trains a binary SVM for anytwo classes of data and obtains a decision function. Thus, for a k-class problem, there are $\frac{k(k-1)}{2}$ decision functions. A voting strategy is used where the testing point is designated to be in a class with the maximum number of votes.

1. One Against All multi-class SVM

The data used for a multi-classification task are generated by the lines below which use the function function `dataset`.

```
n=20;
sigma=1;
x1=sigma*randn(1,n)-1.4;
x2=sigma*randn(1,n)+0;
x3=sigma*randn(1,n)+2;

y1=sigma*randn(1,n)-1.4;
y2=sigma*randn(1,n)+2;
y3=sigma*randn(1,n)-1.4;

xapp=[x1 x2 x3;y1 y2 y3]';
yapp=[1*ones(1,n) 2*ones(1,n) 3*ones(1,n)]';
```

Here the training set contains three different classes and there are 20 samples in each class. The decision function is learned +via this segment of code lines:

```
c = 1000;
lambda = 1e-7;
kerneloption= 2;
kernel='gaussian';
verbose = 1;

%---------------------One Against All algorithms
    ----------------
nbclass=3;
[xsup,w,b,nbsv]=svmmulticlassoneagainstall(xapp,yapp,
    nbclass,c,lambda,kernel,kerneloption,verbose);
```

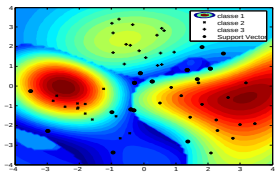

Figure 12: Illustration of multi-class SVM using one against all method

The OAL algorithm is used to obtain the final decision function. Codes in the following lines are used to evaluate its performence.

```
[xtesta1,xtesta2]=meshgrid([-4:0.1:4],[-4:0.1:4]);
[na,nb]=size(xtesta1);
xtest1=reshape(xtesta1,1,na*nb);
xtest2=reshape(xtesta2,1,na*nb);
xtest=[xtest1;xtest2]';
[ypred,maxi] = svmmultival(xtest,xsup,w,b,nbsv,kernel,
   kerneloption);
ypredmat=reshape(ypred,na,nb);
```

2. One Against One multi-class SVM

The data using for the multi-classification task are generated by the lines below which use the function function `dataset`.

```
n=100;
sigma=0.5;
x1=sigma*randn(1,n)-1.4;
x2=sigma*randn(1,n)+0;
x3=sigma*randn(1,n)+2;

y1=sigma*randn(1,n)-1.4;
y2=sigma*randn(1,n)+2;
y3=sigma*randn(1,n)-1.4;

xapp=[x1 x2 x3;y1 y2 y3]';
yapp=[1*ones(1,n) 2*ones(1,n) 3*ones(1,n)]';
nbclass=3;
[n1, n2]=size(xapp);
```

Here the training set contains three different classes and there are 100 samples in each class. The decision function is learned +via this segment of code lines:

```
%   Learning and Learning Parameters
c = 1000;
```

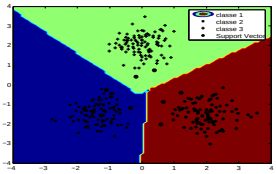

Figure 13: Illustration of multi-class SVM using one against one method

```
lambda = 1e-7;
kerneloption= 1;
kernel='poly';
verbose = 0;

%----------------------One Against One algorithms
    -----------------
nbclass=3;
%
%[xsup,w,b,nbsv,classifier,pos]=
    svmmulticlassoneagainstone(xapp,yapp,nbclass,c,lambda,
    kernel,kerneloption,verbose);
%kerneloptionm.matrix=svmkernel(xapp,kernel,kerneloption)
    ;
[xsup,w,b,nbsv,classifier,pos]=svmmulticlassoneagainstone
    ([],yapp,nbclass,c,lambda,'numerical',kerneloptionm,
    verbose);
```

The following codes are introduced to establish the test data and evaluate performance of the algorithm

```
[xtesta1,xtesta2]=meshgrid([-4:0.1:4],[-4:0.1:4]);
 [na,nb]=size(xtesta1);
 xtest1=reshape(xtesta1,1,na*nb);
 xtest2=reshape(xtesta2,1,na*nb);
 xtest=[xtest1;xtest2]';

% [ypred,maxi] = svmmultivaloneagainstone(xtest,xsup,w,b,
    nbsv,kernel,kerneloption);
kerneloptionm.matrix=svmkernel(xtest,kernel,kerneloption,
    xapp(pos,:));
 [ypred,maxi] = svmmultivaloneagainstone([],[],w,b,nbsv,'
     numerical',kerneloptionm);
```

4 Unsupervised Learning

4.1 Regularization Networks (RN)

4.1.1 Principle

Regularization Networks derived from regularization theory. It is a family of feed-forward neural networks with one hidden layer. 'Learn from examples' is its typical character. Given a set of data samples $\{(x_i, y_i)\}_{i=1}^n$ Our goal is to recover the unknown function or find the best estimate of it.
Unlike in SVM, RN try to optimize some different *smoothness* criterium for the function in input space. Thus we get

$$R_{reg}[f] := R_{emp}[f] + \frac{\lambda}{2}\|Pf\|^2 \tag{16}$$

Here P denotes a regularization operator which is positive semidefinite It realizes the mapping from the Hilbert space H of functions f under consideration to a dot product space D such that the expression $< Pf.Pg >$ is well defined for $f, g \in H$. Using an expansio of f in terms of some symmetric function $\kappa(x_i, x_j)$, we achieves:

$$f(x) = \sum_{i=1}^{n} \alpha_i \kappa(x_i, x) + b \tag{17}$$

Here the κ needs not fulfill Mercer's condition. Similar to the one for SVs, equation 16 leads to a quadratic programming problem. By computing Wolfe's dual and using

$$D_{ij} :=< (Pk)(x_i, .).(Pk)(x_j, .) >$$

we get $\alpha = D^{-1}K(\beta - \beta^*)$ with β, β^* being the solution of

$$\begin{array}{l} \min \frac{1}{2}(\beta^* - \beta)^T K D^{-1} K(\beta^* - \beta) - (\beta^* - \beta)^T y - \epsilon \sum_{i=1}^{n}(\beta_i + \beta_i^*) \\ \text{s.t.} \sum_{i=1}^{n}(\beta_i - \beta_i^*) = 0 \end{array} \tag{18}$$

But this setting of the problem does not preserve sparsity in terms of the coefficients.

4.1.2 Solving the RN problem using the toolbox

1. Classification with regularization networks

The Checker Data are used for the example:

```
nbapp=600;
nbtest=0;
[xapp,yapp]=dataset('Checkers',nbapp,nbtest);
[xtest1 xtest2]  = meshgrid([-2:0.1:2]);
nn = length(xtest1);
xtest = [reshape(xtest1 ,nn*nn,1) reshape(xtest2 ,nn*nn
   ,1)];
```

Learning parameters are set by the codes below:

```
lambda = 1;
kernel='gaussian';
kerneloption=0.2;
```

The line below returns the parameters c and d that equation 16 is minimized.

```
[c,d]=rncalc(xapp,yapp,kernel,kerneloption,lambda);
```

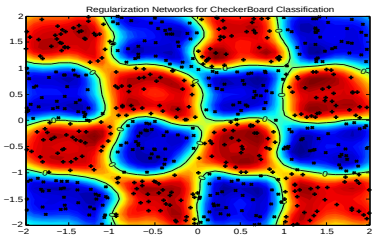

Figure 14: Illustration of RN for CheckerBoard Classifiation

Then we can obtain the output of RN:

```
ypred=rnval(xapp,xtest,kernel,kerneloption,c,d);
```

Figure 14 shows results of Regulation Networks for CheckerBoard Classification.

2. Semiparametric classification with regularization networks

Training set and testing set are generated by the codes below:

```
nbapp=300;
nbtest=25;
[xapp,yapp]=dataset('CosExp',nbapp,nbtest);
[xtest1 xtest2]  = meshgrid(linspace(0,4,nbtest),linspace
   (-1,1,nbtest));
```

Learning parameters are set via the following line:

```
lambda = 1;
kernel='gaussian';
kerneloption=0.1;
T=phispan(xapp,'sin2d');
```

For semiparametric learning, the span matrix $T_{ij} = \phi_j(x_i)$ is given in form of T. Then we get the parameters c and d through the line below:

```
[c,d]=rncalc(xapp,yapp,kernel,kerneloption,lambda,T);
```

Finally, the predict value of RN is obtained:

```
ypred=rnval(xapp,xtest,kernel,kerneloption,c,d,T);
ypred = reshape(ypred,nn,nn);
```

Figure 15 shows results of semiparametric RN classification in CosExp dataset.

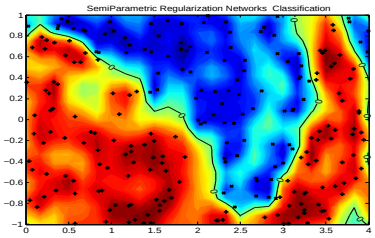

Figure 15: Illustration of Semiparametric RN Classification

4.2 Kernel Principle Component Analysis (Kernel PCA)

4.2.1 Principle

Standard PCA is used to find a new set of axes (the principle components of the data) which hopefully better describes the data for some particular purpose. Kernel principal component analysis (kernel PCA) is an extension of principal component analysis (PCA) using kernel methods. Using a kernel, the originally linear operations of PCA are done in a reproducing kernel Hilbert space with a non-linear mapping.
First, we choose a transformation $\phi(x)$, We apply these transformations to the data and arrive at a new sample covariance matrix:

$$C = \frac{1}{n}\sum_{i=1}^{n}\phi(x_i)\phi(x_i)^T \tag{19}$$

Noting that the transformations must conserve the assumption that the data is centered over the rigion.
To find the first principle component, we need to solve $\lambda\nu = CV$. Substituting for C, this equation becomes:

$$\frac{1}{n}\sum_{i=1}^{n}(\phi(x_i)(\phi^T(x_i)\nu) = \lambda\nu \tag{20}$$

This is the function we want to solve, Since $\lambda \neq 0$, ν must be in the spac of $\phi(x_i)$, i.e. ν can be written by some linear combination of $\phi(x_i)$, namely, $\nu = \sum_i \alpha_i\phi(x_i)$. Substituting for ν and multiplied by $\phi(x_k)$, $k = 1, \cdots, n$ at both sides of equation 20, this give us:

$$\frac{1}{n}\sum_{j}\alpha_j\sum_{i=1}^{n}(\phi^T(x_k)\phi(x_i))(\phi^T(x_i)\phi(x_j)) = \lambda\sum_{i=1}^{n}\alpha_i\phi^T(x_k)\phi(x_i) \tag{21}$$

Define the Gram Matrix

$$K_{ij} = K(x_i, x_j) = \phi^T(x_i)\phi(x_j)$$

Using the define of κ, we can rewrite the equation 20 as:

$$\lambda\kappa\alpha = \frac{1}{n}K^2\alpha \tag{22}$$

Then we have to solve an eigenvalue problem on the Gram matrix. Now to extract a principle component we simply take

$$y_i = \nu^t x$$

For all i, where ν is the eigenvector corresponding to that principle component, noting that

$$y_i = \sum_j \alpha_i(\phi^T(x_j)\phi(x_i)) = \sum_j \alpha_j K(x_i, x_j) \tag{23}$$

4.2.2 Solving the problem using the toolbox

We utilize 3 examples to make the description more detail.

1. Example of Kernel PCA on artificial data

We use the following lines to establish the training set and testing set for the example:

```
x1=0.2*randn(100,1)+0.5;
x2=x1+0.2*randn(100,1)+0.2;
x=[x1 x2];
y1=1*randn(100,1)+0.5;
y2=-y1.^2+0.05*randn(100,1)+0.2;
y=[y1 y2];
xpca=[x;y];


xpca=randn(100,2)*[0.2 0.001; 0.001 0.05];


vect=[-1:0.05:1];
nv=length(vect);
[auxx,auxy]=meshgrid(vect);
xtest=[reshape(auxx,nv*nv,1) reshape(auxy,nv*nv,1)];
```

The eigenvalue and eigenvector are learned via this segment of code lines:

```
kernel='poly';
kerneloption=1;
max_eigvec=8;
[eigvect,eigval]=kernelpca(xpca,kernel,kerneloption);
max_eigvec=min([length(eigval) max_eigvec]);
feature=kernelpcaproj(xpca,xtest,eigvect,kernel,
   kerneloption,1:max_eigvec);
```

Figure 16 shows results of Kernel PCA: Data and IsoValues on the different eigenvectors.

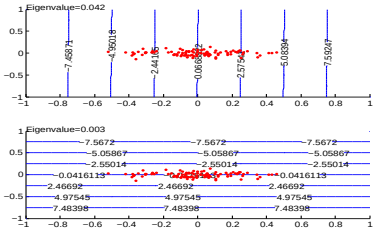

Figure 16: Illustration of Kernel PCA

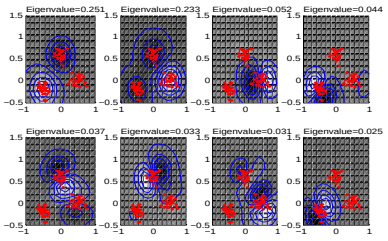

Figure 17: Illustration of Kernel PCA and Kernelpcaproj routine

2. A toy example that make use of Kernel PCA and Kernelpcaproj routine

The data set used in this example is composed of three clusters. Parameters that used to generate the data set is set:

```
rbf_var = 0.1;
xnum = 4;
ynum = 2;
max_ev = xnum*ynum;
% (extract features from the first <max_ev> Eigenvectors)
x_test_num = 15;
y_test_num = 15;
cluster_pos = [-0.5 -0.2; 0 0.6; 0.5 0];
cluster_size = 30;
```

The training set and testing set are generated via the codes below:

```
num_clusters = size(cluster_pos,1);
train_num = num_clusters*cluster_size;
patterns = zeros(train_num, 2);
range = 1;
randn('seed', 0);
for i=1:num_clusters,
  patterns((i-1)*cluster_size+1:i*cluster_size,1) =
     cluster_pos(i,1)+0.1*randn(cluster_size,1);
  patterns((i-1)*cluster_size+1:i*cluster_size,2) =
     cluster_pos(i,2)+0.1*randn(cluster_size,1);
end
test_num = x_test_num*y_test_num;
x_range = -range:(2*range/(x_test_num - 1)):range;
y_offset = 0.5;
y_range = -range+ y_offset:(2*range/(y_test_num - 1)):
   range+ y_offset;
[xs, ys] = meshgrid(x_range, y_range);
test_patterns(:, 1) = xs(:);
test_patterns(:, 2) = ys(:);
```

Then Kernel PCA and Kernelpcaroj are introduced to deal with the problem:

```
[evecs,evals]=kernelpca(patterns,kernel,kerneloption);
test_features=kernelpcaproj(patterns,test_patterns,evecs,
   kernel,kerneloption,1:max_ev);
```

Figure 17 shows results of a toy example that make use of Kernel PCA and Kernelpcaproj routine.

3. Example of kernel PCA as prepocessing stage for a linear (or nonlinear) SVM classifier

This method is also called multilayers SVM. The data for this example are generated by the following codes:

```
N=200;
sigma=1;
[xapp yapp]=dataset('clowns',N,N,sigma);
[x,y]=meshgrid(-4:0.1:4);
```

```
nx=size(x,1);
xtest=[reshape(x,nx*nx,1) reshape(y,nx*nx,1)];
indplus=find(yapp==1);
indmoins=find(yapp==-1);
```

Then the data are pre-treated by Kernel PCA method via the codes below:

```
kernel='gaussian';
kerneloption=1;
max_eigvec=500;
[eigvect,eigval]=kernelpca(xapp,kernel,kerneloption);
max_eigvec=min([length(eigval) max_eigvec]);
appfeature=kernelpcaproj(xapp,xapp,eigvect,kernel,
   kerneloption,1:max_eigvec);
testfeature=kernelpcaproj(xapp,xtest,eigvect,kernel,
   kerneloption,1:max_eigvec);
```

A linear classifier is achieved through the following lines:

```
c = 1000;
lambda =1e-7;
kerneloption = 1;
kernel='poly';
verbose = 0;
[xsup,w,b,pos]=svmclass(appfeature,yapp,c,lambda,kernel,
   kerneloption,verbose);
```

And a nonlinear classifier is achieved through the codes below:

```
c = 1000;
lambda=1e-7;
kerneloption = 1;
kernel='gaussian';
verbose = 0;
[xsup,w,b,pos]=svmclass(xapp,yapp,c,lambda,kernel,
   kerneloption,verbose);
```

Figure 18 shows results of the multilayers SVM when classifing the 'clowns' data set.

4.3 Linear Discriminant Analysis (LDA)

4.3.1 Principle

Linear Discriminant Analysis (LDA) is used to find the linear combination of features which best separate two or more classes of objects or events. It explicitly attempts to model the difference between the classes of data. LDA explicitly attempts to model the difference between the classes of data. For two classes classification task, LDA approaches the problem by assuming that the probability density functions $p(\overrightarrow{x}|y=1)$ and $p(\overrightarrow{x}|y=0)$ are both normally distributed. It also makes the simplifying homoscedastic assumption (i.e. that the class covariances are identical, so $\sum_{y=0} = \sum_{y=1} = \sum$) and that the covariances have full rank.
Let's assume m clusters and n samples:
C: total covariance matrix
G: convariance matrix of the centers
C: total covariance matrix
G: convariance matrix of the centers

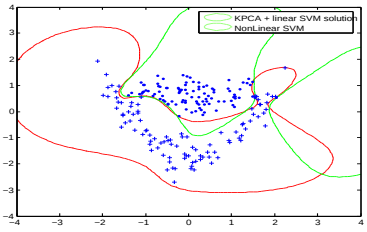

Figure 18: Illustration of Kernel PCA and Kernelpcaproj routine

$\overrightarrow{v}_i$: ith discriminant axis ($i = 1, \cdots, m-1$)

LDA maximizes the variance ratio:

$$\lambda_i = \frac{Interclass \quad variance}{Total \quad variance} \rightarrow \lambda_i = \frac{\overrightarrow{v}_i^T G \overrightarrow{v}_i}{\overrightarrow{v}_i^T C \overrightarrow{v}_i} \tag{24}$$

The solution is based on an eigen system:

$$\lambda_i \overrightarrow{v}_i = C^{-1} G \overrightarrow{v}_i \tag{25}$$

The eigen vectors are linear comhinations of the learning samples:

$$\overrightarrow{v}_i = \sum_{i=1}^{n} \alpha_{ij} \overrightarrow{x}_j \tag{26}$$

4.4 Solving the LDA problems using the toolbox

4.5 Smallest sphere

4.5.1 Princople

How to find the smallest sphere that would include all data?
Training set is $x_1, \cdots, x_n$, seeking the smallest sphere that would include all points lead to the QP problem:

$$\begin{array}{ll} \max W(\alpha) = \sum_{i=1}^{n} \alpha_i \kappa(x_i, x_i) - \sum_{i,j=1}^{n} \alpha_i \alpha_j \kappa(x_i, x_i) & \\ \textbf{s.t.} \sum_{i=1}^{n} \alpha_i = 1 & \\ \alpha_i \geq 0 & i = 1, \cdots, n \end{array} \tag{27}$$

The radius of the smallest sphere is:

$$r^* = \sqrt{W(\alpha^*)}$$

4.5.2 Sloving the problem using the toolbox

The two dimension data here are generated by:

```
N=2;
xapp=[1 1; 2 2];
N=100;
rho=4;
moy=[0 0];
xapp=randn(N,2) + ones(N,1)*moy;
```

Then the outliers are abandoned:

```
mod2=sum((xapp-ones(N,1)*moy).^2,2);
xapp(find(mod2>rho),:)=[];
```

Learning parameters are set via the following lines:

```
kernel='poly';


C=0.001;
verbose=1;
kerneloption=1;
```

In this example, we have to take C into account when changing the kernel:

```
K=svmkernel(xapp,kernel,kerneloption);
kerneloption.matrix=K+1/C*eye(size(xapp,1));
```

Finally, we get the radius and the center of the sphere when launching the line:

```
[beta,r2,pos]= r2smallestsphere([],kernel,kerneloption);
```

Figure 19 shows the results of smallest sphere, and you can get the radius is 31.4717 in the command window.

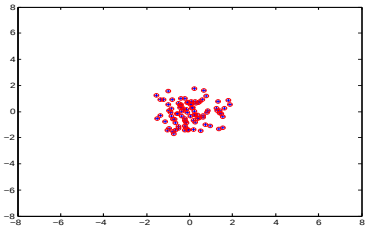

Figure 19: Illustration of solution of smallest sphere

5 Regression Problem

5.1 Principle

We introduce the loss function to solve the regression problem. In ϵ-SV regression task, out goal is to find a function $f(x)$ that has at most ϵ deviation from the actually obtained targets y_i for all the training data, and at the same time, is as flat as possible. In other words, we do not care about errors as long as they are less than ϵ, but will not accept any deviation larger than this.
The decision function for regression taking the form:

$$f(x) = \langle w, \phi(x) \rangle + b \tag{28}$$

Flatness in the case of equation 28 means that one seeks small w. So we can write this problem as a convex optimization problem by requiring:

$$\begin{array}{lll} \min & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}(\xi_i + \xi_i^*) & \\ \text{s.t.} & y_i - \langle w, \phi(x)\rangle - b \leq \epsilon + \xi_i & i = 1, \cdots, n \\ & \langle w, \phi(x)\rangle + b - y_i \leq \epsilon + \xi_i^* & i = 1, \cdots, n \\ & \xi_i, \xi_i^* \geq 0 & i = 1, \cdots, n \end{array} \tag{29}$$

The so-called $\epsilon-$insensitive loss function $|\xi_i|_\epsilon$ is described by:

$$|\xi|_\epsilon := \max(0, |y_i - f(x_i)| - \epsilon) \tag{30}$$

Similar with Support vector classification, we can construct a Lagrange function to solve the optimization problem. Finally, we can obtain:

$$w = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\phi(x_i) \tag{31}$$

Therefore:

$$f(x) = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\left\langle \phi(x_i), \phi(x)\right\rangle + b = \sum_{i=1}^{n}(\alpha_i - \alpha_i^*)\kappa(x_i, x) + b \tag{32}$$

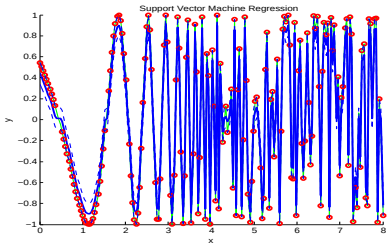

Figure 20: Illustration of Support Vector Regression for 1D data

5.2 Solving the regression problem using the toolbox

1. 1D Support vector regression problem

The data for this example are generated by the codes below:

```
n=200;
x=linspace(0,8,n)';
xx=linspace(0,8,n)';
xi=x;
yi=cos(exp(x));
y=cos(exp(xx));
```

Learning parameters are set via the follwing lines:

```
C = 10;  lambda = 0.000001;
epsilon = .1;
kerneloption = 0.01;
kernel='gaussian';
verbose=1;
[xsup,ysup,w,w0] = svmreg(xi,yi,C,epsilon,kernel,
   kerneloption,lambda,verbose);
```

Figure 20 shows results of Support Vector Regression for 1D data, the blue line denotes the final decision function, and the red points denote support vectors.

2. 2D Support vector regression problem

The data for this example are generated by the codes below:

```
N=100;
x1=2*randn(N,1);
x2=2*randn(N,1);
```

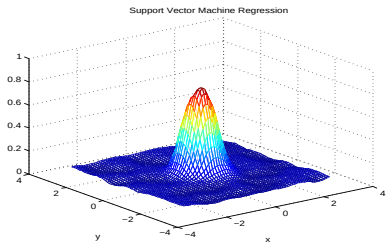

Figure 21: Illustration of Support Vector Regression for 2D data

```
y=exp(-(x1.^2+x2.^2)*2);
x=[x1 x2];
```

Learning parameters are set via the follwing lines:

```
C = 1000;
lambda = 1e-7;
epsilon = .05;
kerneloption = 0.30;
kernel='gaussian';
verbose=1;
[xsup,ysup,w,w0] = svmreg(x,y,C,epsilon,kernel,
   kerneloption,lambda,verbose);
```

Figure 21 shows result of Gaussian Support vector regression for 2D data.

3. Large scale Support vector regression problem

The following codes return data for this example:

```
n=600;
x=linspace(0,5,n)';
xx=linspace(0,5,n)';
xi=x;
yi=cos(exp(x));
y=cos(exp(xx));
```

Learning parameters are set via the lines below:

```
C = 10;  lambda = 0.000001;
epsilon = .1;
kerneloption = 0.4;
```

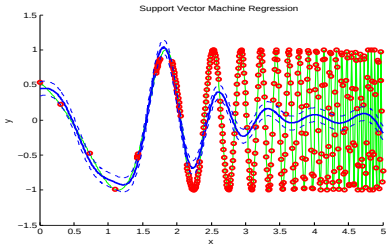

Figure 22: Illustration of Large Scale Support Vector Regression

```
kernel='gaussian';
verbose=1;
qpsize=100;
[xsup,ysup,w,w0] = svmregls(xi,yi,C,epsilon,kernel,
   kerneloption,lambda,verbose,qpsize);
```

Here the chunking method is adopted to solve the large scale regression problem, and the size of QP method is 100.

Figure 22 shows results of large scale support vector regression. In this figure, width of the tube around the final decision function is $\epsilon/2$.

5.3 Linear Programming Support Vector Regression

The decision function in the linear programming SVM for regression is as follows:

$$f(x) = \sum_{i=1}^{n} \alpha_i \kappa(x, x_i) + b \tag{33}$$

where α_i take on real values.
We consider minimizing:

$$\min L(\alpha, \xi) = \sum_{i=1}^{n} (\alpha_i + \alpha_i^*) + C \sum_{i=1}^{n} (\xi_i + \xi_i^*) \tag{34}$$

Subject to:

$$\begin{array}{l} y_i - \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \kappa(x_j, x_i) - b \leq \epsilon + \xi_i \\ \sum_{i=1}^{n} (\alpha_i - \alpha_i^*) \kappa(x_j, x_i) + b - y_i \leq \epsilon + \xi_i^* \\ \alpha_i, \alpha_i^*, \xi_i, \xi_i^* \geq 0 \end{array}$$

where ξ_i are positive slack variables and C is a margin parameter.
The data for the example was generated by the codes below:

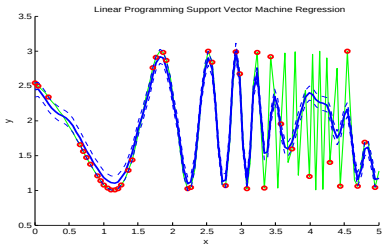

Figure 23: Illustration of Linear Programming Support Vector Regression

```
n=100;
x=linspace(0,5,n)';
xx=linspace(0,5,n)';
xi=x;
yi=cos(exp(x))+2;
y=cos(exp(xx))+2;
```

Learning parameters are set via the lines below:

```
C = 10;  lambda = 0.000001;
epsilon = .1;
kerneloption = 0.1;
kernel='gaussian';
verbose=1;
[xsup,ysup,w,w0] = LPsvmreg(xi,yi,C,epsilon,lambda,kernel
    ,kerneloption,verbose);
```

Figure 23 shows result of Linear Programming SVM for regression task.Compared with standard SVM, it has less support vectors.